## Config.xml Guide

The config file is the heart of your ColdBox application. It contains the initialization variables for your application and extra information used by the ColdBox plugins. Below is an overview of every section of the file. **All the data in the config.xml.cfm will be placed in the configStruct. If you would like to reload your settings, you will need to reinitialize the framework by using the fwreinit=1 URL action.**

**Note About Security:** Your config.xml file is actually a cfm template. I did this in order to protect the download of the file with the use of an Application.cfm template (Thanks to Raymond Camden). However, for added security I also use an apache acces file: .htaccess. You will find the .htaccess file on the distribution archive for the config directory. This is used by Apache for executing directory security.

(**Read more on .htaccess)** .

In IIS, you will have to configure the directory's security from the IIS Manager.

(**Good article on securing IIS)**

### <Settings>

This section is for ColdBox settings. These are pre-defined and you cannot rename them or ommit them since they are validated against its XML Schema, please remember that **XML IS CASE-SENSATIVE** .

| Setting | Description | Type |
|---|---|---|
| **AppName:** | The unique name of your application | String |
| **AppCFMXMapping** | The Coldfusion Mapping for your application. Leave blank if the application is in the root of your webserver. | String |
| **DebugMode** | Enable/Disable ColdBox debug mode. | Boolean |
| **DebugPassword** | The password you would like to use to go into debugmode. You will need to pass this string as a URL param combined with the DebugMode to use. Look at examples below | String |
| **DumpVarActive** | Enable/Disable the use of the URL action to dump variables. | Boolean |
| **ColdfusionLogging** | Enable/Disable Coldfusion Error Logs. | Boolean |
| **DefaultEvent:** | The name of the event handler for the default event to run: ehGeneral.dspHome | String |
| **RequestStartHandler:** | The name of the onRequestStart handler to run: ehGeneral.onRequestStart, leave blank if not used. | String |
| **RequestEndHandler:** | The name of the onRequestEnd handler to run: ehGeneral.onRequestEnd, leave blank if not used | String |

| | | |
|---|---|---|
| **ApplicationStartHandler** | The name of the onApplicationStart handler to run: ehGeneral.onAppStart, leave blank if not used. To trigger again this handler, you must reinitialize the framework: fwreinit=1 | String |
| **OwnerEmail:** | The email that will be used to send all email communications. | String |
| **EnableBugReports:** | Enable/Disable the emailing of bug reports | Boolean |
| **UDFLibraryFile** | The location of your UDF library if in use, else leave blank. ColdBox will first look in your includes directory, so you can just place the name of the UDF here, or the full path you write including CFMX Mappings. Ex: /ColdBoxSamples/includes/udf.cfm | String |
| **ExceptionHandler** | The custom exception handler to run on all framework exceptions. You decide what to do. | String |
| **CustomErrorTemplate** | ColdBox comes with its own error template. However, if you wish to customize your errors, which you should, then just place the location of your custom error template. For example: includes/errorpage.cfm or /mymapping/templates/error.cfm Then in order to retrieve the error, you will need to get the Exception Bean from the request collection: getValue("ExceptionBean"). You can then use this bean to render your error page. Please look at the API to get a better understanding of the bean. | String |
| **MessageboxStyleClass** | The CSS class name to be used with the messagebox plugin. If left blank, ColdBox will use its internal CSS class. | String |
| **HandlersIndexAutoReload** | This is a flag mostly used during development. ColdBox onApplication Start will read your handlers directory and store the names of the available handlers. When requests are made and handlers get instantiated, they are instantiated using the internal syntax. Thus, if you are developing and are adding handlers, with this flag set to TRUE, then ColdBox will reload the list. Else, you will have to manually reload the structures using the **fwreinit=1 url action** | Boolean |
| **ConfigAutoReload** | This is a flag mostly used during development. It will reload your config.xml settings on every request. Else you will have to manually reload the structures using fwreinit=1 | Boolean |

**DebugMode example:**
In order to enter/leave ColdBox debugmode, you will need to append some URL parameters in order to do this. The url below will activate debugmode:

*http://apppath/index.cfm?debugmode=true&debugpass=ColdBox*

This url tells ColdBox to enable debugmode and use the debugpass variable to test against the config.xml.cfm DebugPassword settting. If you do not want to assign a debug password, you can leave the setting blank.
The url below will deactivate debugmode.

*http://apppath/index.cfm?debugmode=false&debugpass=ColdBox.*

**<YourSettings>**

This section is used only if you want to specify your own settings for your application. Like for example your datasource name, your own email address, etc. You can then retrieve them using the ***getSetting()*** method. Example:

```
<YourSettings>
   <Setting name="myurl" value="http://myurl.com" />
   <Setting name="mysetting" value="myvalue" />
</YourSettings>
```

**<MailServerSettings>**

This section is provided to bypass the Coldfusion Administrator's mail settings. For example: for hosted environments where you do not have access to the ColdFusion Administrator. You can then retrieve them using the ***getSetting()*** method

```
<MailServerSettings>
   <MailServer>mail.mymailserver.com</MailServer>
   <MailUsername>luismajano</MailUsername>
   <MailPassword>PASSWORD</MailPassword>
</MailServerSettings>
```

**<BugTracerReports>**

ColdBox can send multiple users bug reports. You define all the email addresses that will receive these mail reports. However, in order for the bug reports to be active, you need to set the **EnableBugReports** setting to true. You can then retrieve them using the ***getSetting()*** method

```
<BugTracerReports>
   <BugEmail>lmajano@gmail.com</BugEmail>
   <BugEmail>someone@mail.com</BugEmail>
</BugTracerReports>
```

**<DevEnvironments>**

This section is used to keep track of your environment. You can list as many url's as you want, ColdBox will try to match at least one. If it does then it will set the ENVIRONMENT variable to DEVELOPMENT, else to PRODUCTION. You can then retrieve them using the ***getSetting()*** method

```
<DevEnvironments>
  <url>dev</url>
  <url>lmajano</url>
</DevEnvironments>
```

**<WebServices>**

You may use this section to list all your webservices that your application can use instead of registering them in the Coldfusion Administrator. This becomes useful, since you can refresh their stubs using the webservices API. Another nice feature, is that you can declare for every webservice a development url and/or a production url. You can then retrieve them using the ***getSetting()*** method

```
<WebServices>
  <WebService name="DistributionWS"
  URL="http://ColdBox.luismajano.com/ColdBox.cfc?wsdl"
  DevURL="http://dev.com/ColdBox.cfc?wsdl" />
  <WebService name="GoogleWS" URL="http://ws.google.com/ws?wsdl"/>
</WebServices>
```

**<Layouts>**

This is a very important setting for your ColdBox application. You will need to fill out a mandatory setting which is the element. This element tells ColdBox to always use this Layout unless specifically specified. You can then retrieve them using the ***getSetting()*** method

**For example:** Your application only uses one layout, then this section would look like this:

```
<Layouts>
  <DefaultLayout>Layout.Main.cfm</DefaultLayout>
<Layouts>
```

However, if your application has some views that need a different layout, then you need to define them here. You can also change a view's layout programmatically by using the **SetLayout({layout_name})** method. So you define a Layout first, with a file and name attribute. You will then proceed to create **<View>** elements with the name of the view. Do not use the .cfm extension, ColdBox applies it automatically. You can then retrieve them using the ***getSetting()*** method.

```
<Layouts>
  <DefaultLayout>Layout.Main.cfm</DefaultLayout>
  <Layout file="Layout.Login.cfm" name="login">
    <View>vwLogin</View>
  </Layout>
  <Layout file="Layout.Open.cfm" name="open">
    <View>vwLuis</View>
    <View>vwPopup</View>
  </Layout>
<Layouts>
```

**<i18N>**

This element is used to define a default resource bundle, a default java standard locale and the storage for the locale in your ColdBox application. This will be used to activate ColdBox's Internationalization features. ColdBox will read the defined resource bundle according to the set locale, parse it and store it in an internal **Application** variable. Then from the event handlers, layouts or views you can just use the ***getResource("Key")*** method to get keys from the bundle structure. ex: ***getResource("cancelbutton")***

For now, ColdBox can only use i18N via properties files. The database version is on the works. The options for LocaleStorage are session or client. This is just the scope where ColdBox will place the ***DefaultLocale*** variable in. You can manipulate this variable and more through the ***i18n*** plugin. Please look at the API.

```
<i18N>
  <DefaultResourceBundle>includes/main</DefaultResourceBundle>
  <DefaultLocale>en_US</DefaultLocale>
  <LocaleStorage>session</LocaleStorage>
<i18N>
```